**Multi Threading**

**Information about multithreading:-**

1. The earlier days the computer’s memory is occupied only one program after completion of one program it is possible to execute another program is called uni programming.
2. Whenever one program execution is completed then only second program execution will be started such type of execution is called co operative execution, this execution we are having lot of disadvantages.
   * 1. Most of the times memory will be wasted.
     2. CPU utilization will be reduced because only program allow executing at a time.
     3. The program queue is developed on the basis co operative execution

**To overcome above problem a new programming style will be introduced is called multiprogramming.**

* 1. Multiprogramming means executing the more than one program at a time.
  2. All these programs are controlled by the CPU scheduler.
  3. CPU scheduler will allocate a particular time period for each and every program.
  4. Executing several programs simultaneously is called multiprogramming.
  5. In multiprogramming a program can be entered in different states.
     + 1. Ready state.
       2. Running state.
       3. Waiting state.
  6. Multiprogramming mainly focuses on the number of programs.

**Advantages of multiprogramming:-**

1. CPU utilization will be increased.
2. Execution speed will be increased and response time will be decreased.
3. CPU resources are not wasted.

**Thread:-**

1. Thread is nothing but separate path of sequential execution.
2. The independent execution technical name is called thread.
3. Whenever different parts of the program executed simultaneously that each and every part is called thread.
4. The thread is light weight process because whenever we are creating thread it is not occupying the separate memory it uses the same memory. Whenever the memory is shared means it is not consuming more memory.
5. Executing more than one thread a time is called multithreading.

**Single threaded model:-** class Test

{ begins

public static void main(String[] args)

{

System.out.println("Hello World!");

System.out.println("hi cg"); body

System.out.println("hello ");

}

} end

In the above program only one thread is available is called main thread to know the name of the thread we have to execute the fallowing code.

class Test

{

public static void main(String[] args)

{

System.out.println("Hello World!");

Thread t=Thread.currentThread();

System.out.println("currrent thread information is : "+t);//[main,5,main]

System.out.println("currrent thread priority is : "+t.getPriority());//5

System.out.println("currrent thread name is : "+t.getName());

System.out.println("hi ");

System.out.println("hello CG");

}

}

**In the above program only one thread is available name of that thread is main thread.**

**Multithreaded model:-**

Starts

Starts

starts

Thread may switch or exchange data/result.

Main thread

Thread

A

Thread B

Thread C

**The main important application areas of the multithreading are**

1. Developing video games
2. Implementing multimedia graphics.
3. Developing animations

**There are two different ways to create a thread is available**

* 1. Create class that extending standered java.lang.Thread Class
  2. Create class that Implementing java.lang.Runnable interface

**Creation of threads in java**

extends

implements

Thread

MyThread

Runnable

MyClass

Thread

**(a)Objects are threads (b)objects with run() body**

**First approach to create thread extending Thread class:-**

**Step 1:-**

**Creates a class that is extend by Thread classes and override the run() method** class MyThread extends Thread

{

public void run()

{

System.out.println("business logic of the thread");

System.out.println("body of the thread");

}

};

**Step 2:-**

**Create a Thread object**

MyThread t=new MyThread();

**Step 3:-**

**Starts the execution of a thread.**

t.start();

**In this approach take one user defined class class that is extending Thread class .**

**Ex:-**

class MyThread extends Thread

{

public void run()

{

System.out.println("Hello");

System.out.println("body of the thread");

}

};

class ThreadDemo

{

public static void main(String[] args)

{

MyThread t=new MyThread(); t.start();

}

}

Note :-

1. Whenever we are calling t.start() method the JVM search for the start() in the MyThread class but the start() method is not present in the MyThread class so JVM goes to parent class called Thread class and search for the start() method.
2. In the Thread class start() method is available hence JVM is executing start() method.
3. Whenever the thread class start() that start() is responsible person to call run() method.
4. Finally the run() automatically executed whenever we are calling start() method.
5. Whenever we are giving a chance to the Thread class start() method then only a new thread will be created.

**Life cycle stages are:-**

1. **New**
2. **Ready**
3. **Running state**
4. **Blocked / waiting / non-running mode**
5. **Dead state**

**New :-**

MyThread t=new MyThread();

**Ready :-**

t.start()

**Running state:-**

If thread scheduler allocates CPU for particular thread. Thread goes to running state

The Thread is running state means the run() is executed.

**Blocked State:-**

If the running thread got interrupted of goes to sleeping state at that moment it goes to the blocked state.

**Dead State:-**

If the business logic of the project is completed means run() over thread goes dead state.

**Second approach to create thread implementing Runnable interface:-**

**Step 1:-**

**Creates a class that implements Runnable interface.**

class MyClass extends Runnable

{

public void run()

{

System.out.println("Hello");

System.out.println("body of the thread");

}

};

**Step 2:-**

**Creating a object.**

MyClass obj=new MyClass();

**Step 3:-**

**Creates a Thread class object.**

Thread t=new Thread(obj);

**Step 4:-**

**Starts the execution of a thread.**  t.start();

**implementing Runnable interface**

class MyThread implements Runnable

{

public void run()

{

System.out.println("Hello");

System.out.println("body of the thread");

}

}

class ThreadDemo

{

public static void main(String[] args)

{

MyClasss obj=new MyClass(); Thread t=new Thread(obj); t.start();

}

}

**Step 1:-** the Class MyClass implements the Runnable interface and overriding run() method and contains the logic associates with the body of the thread.

**Step 2:-**

Creates the object of implementation class this is not like a first mechanism.

**Step 3 :-**

Creates a generic thread object then pass the MyClass reference variable as a parameter to that object.

**Step 4:-**

As a result of third step 3 a thread object is created in order to execute this thread method we need to class start() method. Then new thread is executed.

**We are having two approaches:-**

**First approach:-**

1. By extending the thread class, the derived class itself is a thread object and it gains full control over the thread life cycle.
2. Another important point is that when extending the Thread class, the sub class cannot extend any other base classes because Java allows only single inheritance.

**if the program needs a full control over the thread life cycle, then extending the Thread class is a good choice.**

**Second approach:-**

1. Implementing the Runnable interface does not give developers any control over the thread itself, as it simply defines the unit of work that will be executed in a thread.
2. By implementing the Runnable interface, the class can still extend other base classes if necessary.

**if the program needs more flexibility of extending other base classes, implementing the Runnable interface would be preferable.**

We are having two approaches two create a thread use any approach based on application requirement.

**Thread life cycle:-**

New

ready

Start()

Running state

If CPU is allocated

sleeping

Sleep()

If time is expired

waiting

Wait()

Notify()

Dead state

If run() completes/

Stop()

**Thread life cycle**

Synchronized: -

• Synchronized modifier is the modifier applicable for methods but not for classes and variables.

• If a method or a block declared as synchronized then at a time only one Thread is allowed to operate on the given object.

• The main advantage of synchronized modifier is we can resolve data inconsistency problems.

• But the main disadvantage of synchronized modifier is it increases the waiting time of the Thread and effects performance of the system. Hence if there is no specific requirement it is never recommended to use.

• The main purpose of this modifier is to reduce the data inconsistence problems.